ANALYZING EMBEDDED DEVICE PROGRAMMING SPECIFICATIONS IN CRITICAL SYSTEMS TO DESIGN PROGRAMMING GUIDELINES

E.S. Korolyuk

esk13@tpu.ru

Scientific Supervisor: Ph.D. Brazovsky K.S.; Language supervisor: Vasilii Morozov, Sr. Lecturer. Dept. of Foreign Languages of the IC, TPU

Software safety is an important characteristic, which indicates the probability of correct program operation under different operating conditions. However, embedded software developers do not always focus on this aspect, especially in critical systems. The present article discusses the international software security standards and certificates. The author studies the source code of the presently discontinued electrosurgical device EHVCh-80 by NPO «NIKOR» and analyzes the issues of validity of choosing the programming language and architecture. The paper provides the algorithm of the program's operation and describes the main tasks performed by the microcontroller. In conclusion, the author uses the results of this software's analysis to offer recommendations to simplify software support and possible options to reduce errors, both in the code and in the architecture of the developed application. The results of this research allow to increase software safety, especially in critical systems.

With improvement of control systems in embedded designs, especially in systems that are critical to safety, it is necessary to ensure the safety and reliability of the designed software. Designers should understand that not only hardware failure (electrical, mechanical, etc.) in a medical device can cause the danger to the patient. Software bugs can also cause serious problems.

There is a number of international standards and security certificates, for example MISRA C [1] or Safety Integrity Level [2]. When using these standards, the safety analysis lies in reducing a potential source of harm and decreasing it to the minimum. It is necessary to consider all steps of device development, from inception to decommissioning.

SIL (Safety Integrity Level) certification addresses system level reliability. Major accidents such as the Bhopal disaster (1984), the Piper Alpha oil rig disaster in the North Sea (1988) and others, have led to the creation of this certification. During the investigation of these events, demands were made by government agencies to check the systems responsible for safety. This led to the development of the international standards ISA S84 [3] and IEC 61508/61511[4] that in turn formed the basis of Russian standards such as GOST R IEC 61508-1-2007 [5]. The value of the level of reliability depends on whether the system will be subjected to more or less stringent requirements.

The MISRA standard is a set of requirements (rules) for software development in the C language in addition to the standard rules.

Both higher-order language and assembler language have their advantages in writing programs for microcontrollers. The main advantage of assembler is fast performance and small code size. On the other hand, there are disadvantages of poor readability and difficulties in maintenance. This adds additional constraints if it is necessary to add new features or change the structure of the program. If used incorrectly, a command to the microcontroller can change data and cause undesired operation of a device. Rigid attachment of a language to a specific platform makes it impossible to perform automatic analysis of code for security.

The reasons for the C language's popularity in embedded systems are simple. The language is standardized, and its constructs are easily associated with machine instructions. The code becomes independent of the microcontroller and it is possible to transfer the program to different platforms.

In addition to syntax errors, there are other problems where the ISO C standard [6] specifies that the implementation may be certain or uncertain. Beginners face difficulties applying this programming language. The solution is to limit the use of some of the language's functions with well-defined opportunities to enhance application performance. To solve these problems, the Motor Industry Software Reliability Association (MISRA) has produced a range of mandatory, required and recommended programming practices [1].

Medical industry produces products that directly affect the patient. Strict safety regulations require additional monitoring throughout the product life cycle during the execution of all procedures. Software must also comply with safety regulations. Product life cycle can be divided into the following stages:

- Research
- Design and development
- Mass production
- Shipping
- Operation
- Repairs
- Disposal

As a result, due to additional requirements for implementation, long periods of time pass between stages in life cycle. The main reason is not only the duration of the certification, but also the developers' need to make profit for further development. Introduction of new functions and improvement of equipment are also a necessity. Even with due reserves, during the design stage a control system (typically a microcontroller or microprocessor) becomes outdated and by the time when an update is necessary, a programmer has to use additional techniques and unconventional programming methods to introduce new features. To analyze the requirements that need to be put forward at the beginning of the development stage, let us consider a discontinued electrosurgical device EHVCh-80 by NPO «NIKOR» company.

ATmega8 microcontroller [7] is the main element of the control system. Its program is written in assembly language in the AVR Studio environment. The microcontroller performs the following tasks:

- ADC reads parameters from sensors attached to the patient. Interrupt occurs at the end of interrupt.
- Power output is regulated by PWM
- When the timer 0 overflows, interrupt is invoked to check the keyboard.
- When the timer 1 overflows, interrupt is invoked and data are checked with the patient. If the device is not in use, its power section is disabled.
- Timer 2 is used for service purposes. Its overflow causes an interrupt and reading/writing of settings data.
- External interrupts are triggered by pressing buttons directly on the instrument.
- General purpose pins are used to control the indicators, the keyboard and the controls and elements of the power part of the device.

The program works according to the following algorithm:

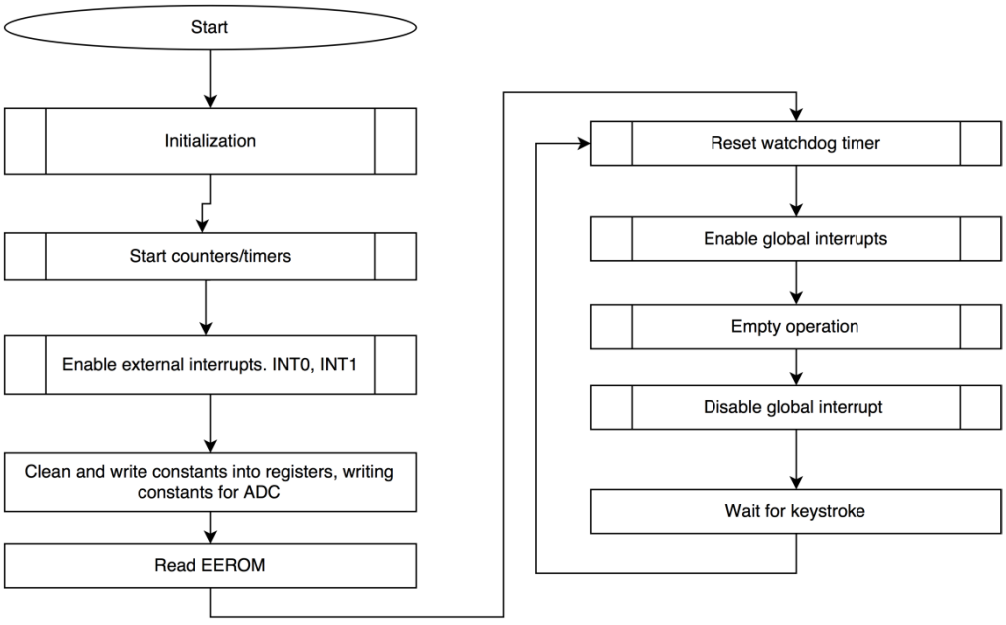

Fig. 1. Algorithm of the EHVCh-80 software.

The basic steps are performed when the global interrupts are enabled. At this time, the interrupt detection conditions are fulfilled. All major procedures are executed, and the flag registers are engaged. In other parts of the code, device operation modes are checked. It is implemented as follows:

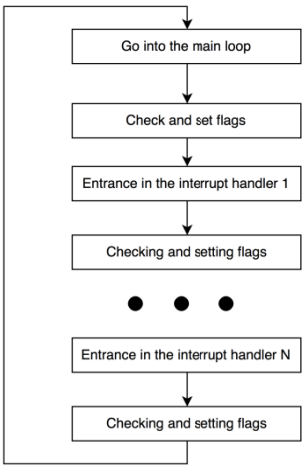

Fig. 2. Algorithm of the interrupt operation in EHVCh-80

This structure style is convenient when the size of the code is small. If you increase its size, there are problems with tracking flags. A flag error may result in enabling a wrong device operation mode. Atmega8 microcontroller has 31 available registers, and the programmer of EHVCh-80 used 4 of the registers, which corresponds to 32 flags. If we change the structure of the program, or require additional software support, we need to know the purpose of each flag.

Conclusion

As a result of analyzing the architecture and the code of the medical device EHVCh-80, it was found that in order to improve the reliability of embedded system software; it is possible to recommend the following:

1. Developers should not put a lot of confidence in their software's correctness. To facilitate understanding of the code it should not be to obfuscated (made confusing).
2. Design must be defensive. Software should contain self-test procedures. Work logs must be kept.
3. The causes of dangerous failures must be prevented. Although in the described device there are independent fuses that cut it off, it is important for the developers to remember about this issue.
4. Do not use bad coding practices. The purpose of each variable must be clear and the functions assigned to it must be described. Never use a single variable to control the device; this is especially important in critical systems.

References

1. MIRA Limited. «MISRA-C: 2004 Guidelines for the use of the C language in critical systems». Edition 2. Warwickshire, UK: MIRA Limited, July 2008 (ISBN 978-0-9524156-4-0).
2. Manual Safety Integrity Level. Andy Ingrey, Patrick Lerévérend, Dr. Andreas Hildebrandt [Electronic resource] – URL: http://goo.gl/IrxoKt (accessed date: 10.03.16).
3. ANSI/ISA 84.00.01-2004 and Existing Safety Instrumented Systems.
4. IEC 61508/61511 Safety Integrity Level [Electronic resource] – URL: http://goo.gl/lWVZOA (accessed date: 10.03.16).
5. GOST R IEC Functional safety of electrical, electronic, programmable electronic safety-related systems. Part 2. Requirements for system . – Moscow.: STANDARTINFORM, 2014
6. ISO/IEC 9899:2011 – Information technology – Programming Languages – C.
7. ATmega8535-16PU Atmel 8-bit Microcontrollers [Electronic resource] – URL: http://www.atmel.com/images/doc2502.pdf (accessed date: 10.03.16).